Peg Solitaire

Digitala Projekt EITF10

Lunds Tekniska Högskola

Maria Krantz, Ylva Nordlund

2010-05-17

Abstract

We have constructed a digital Peg Solitaire game in the course EITF10. The device is built with a 64*128 pixel LCD screen, a keypad consisting of 16 buttons and an AVRMega 16 one-chip microprocessor. Using the keypad the player can navigate the board presented on the screen and choose what pegs to move in order to complete the game. The components were first connected and when we understood how the hardware worked, the programming of the game was made. This document describes the construction as well as the methods used to build and program the device. It also describes problems found during the process and how they were solved.

Innehållsförteckning

Innehåll

1. Inledning

Detta projekt gick ut på att få en inblick i konstruktionsarbete och för att studenterna ska lära sig kopplingen mellan hårdvara och mjukvara. Som studenter från I-programmet var detta ett nytt och spännande ämne.

Vi fick själva välja typ av projekt men skulle utgå från mikroprocessorn ATMega16. Vi valde att lägga till en LCD-skärm på 64*128 pixlar och en knappsats med 16 knappar för att göra ett spel som heter Peg Solitaire.

Nedan följer rapporten med en bskrivning av krav och komponenter, utförandet, problem på vägen samt resultat och diskussion.

2. Krav

Kravspecifikationen var det första som skrevs och styrde senare vad konstruktionen skulle kunna göra. Konstruktionen ska kunna sätta upp ett Peg Solitaire och kunna spela det enligt reglerna för spelet. När bara en kula finns kvar i mitten av brädet är spelet avslutat och man har vunnit.

- Vid start ska brädet ritas upp på LCD-skärmen med fyllda kulor, förutom i mitten där markören ställs.
- Navigation på brädet ska skötas av 4 piltangenter, med en markering på den ruta man befinner sig på.
- Det ska gå att markera den kula man vill flytta på och sedan ska det gå flytta den till en tom plats enligt reglerna för spelet.
- Det ska finnas en RESET knapp för att starta om brädet.

I mån av tid ska följande göras:

- Sätta en tidsbegränsning för hur länge man får spela och införa ett high score register

3. Hårdvara

3.1 ATMega16

ATMega16 är en enchipsprocessor med 16 kb minne med 40 pinnar. Dessa består bland annat av 4 portar som skrivs till med 8 bitar åt gången. Till dessa kopplades skärmen, knappsatsen och en JTAG för kontakt med datorn.

3.2 JTAG

JTAG används för att kunna föra över programvaran vi skrev i AVR Studio till processorn så att programmet skulle kunna köra.

3.3 LCD-skärm

LCD-skärmen bestod av två skärmar om vardera 64*64 pixlar. Vilken skärm som skrevs på styrdes av genom portarna Chipselect 1 och 2. I x-led var skärmarna uppdelade i 8 stycken ”pages” som alltid skrevs alla 8 pixlar på en gång. I y-led skrevs det på skärmen ut pixel för pixel, för varje pixel som skrevs flyttades markeringen en pixel framåt automatiskt.

3.4 Knappsats, 16 knappar

Knapparna hade 8 pinnar, 4 inportar och 4 utportar. Ettor skickades i tur och ordning till inportarna och när en knapp var nedtryckt kunde en etta avläsas från någon av pinnarna på utporten. Knapparna vi använde var följande:

- RST – Reset
- M – Markera
- J - Jump to
- W - Winner test
- Pilar upp, ner, höger och vänster

Siffrorna 2-9 används inte.

3.5 OR-port

Pinnarna på utporten från knappsatsen kopplades till varsina OR-portar. På så sätt skrevs en etta till PC0 om någon av knapparna blev nedtryckta och då kontrollerades vilken knapp som var nedtryckt.

3.6 Resistorer

De fyra utportarna från knappsatsen kopplades även till varsin resistor för att de skulle vara aktiva höga, när ingen knapp var nedtryckt visade de alltså noll.

4. Mjukvara

4.1 Kod

Koden är uppdelad i tre c-filer med respektive h-filer samt en h-fil för utskrift av brädet och bokstäverna.

- **digpiSolitare.c:** Huvudprogrammet och de metoder som rör spelets gång.
- **buttons.c:** Alla metoder som rör knappsatsen och dess kommunikation med processorn.
- **lcdscreen.c:** Alla metoder som rör skärmen och dess kommunikation med processorn.

- **font.h :** Definierar utskriftskommando för brädet och bokstäverna.

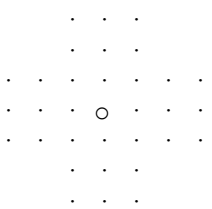
Samtliga filer finns i bilaga 1.

4.2 Polling-avbrott

För att läsa av om en knapp blivit nedtryckt använde vi oss så kallad polling. Knappsatsen hade fyra pinnar dit processorn skickade en etta till en pinne med hjälp av en loop. Vilken pinne som hade en etta bestämde raden på knappsatsen. Efter varje loop lästes pinnen PC0 av för att se om en knapp var intryckt och i så fall bröts loopen och radnumret sparades. Därefter lästes knappsatsens utport av, som bestämde kolonnen. Med värde på både kolonn och rad bestämdes vilken knapp som varit intryckt.

4.3 Spelet

Peg Solitaire är ett brädspel för en spelare som flyttar kulor mellan hål på ett bräde. När spelet börjar ser brädet ut enligt följande, där punkterna respresenterar kulor och o representerar ett tomt hål:



Drag görs genom att man tar en kula och hoppar med den över en annan kula till ett tomt hål. Den kula som blivit överhoppad tas då bort från brädet. Spelet är vunnet när det bara finns en kula kvar på brädet.

För att göra spelet i digital form ritades brädet på skärmen. Vid spelets början finns också en markör på mittenkulan som visar var på skärmen man befinner sig. Spelaren kan sedan navigera med hjälp av knapparna runt på skärmen. Knappen markerad med ”M” används för att markera den kula som man vill flytta. Knappen ”J” används för att flytta den markerade kulan till den

plats på skärmen man befinner sig på. Kulan flyttas bara om draget är möjligt enligt reglerna för spelet. Knappen ”RST” används för att börja om spelet, och knappen ”W” används för testa vad som händer om man lyckas lösa spelet. Då skrivs texten ”YOU WIN” ut på skärmen och man kan sedan trycka ”RST” för att börja om igen.

Se även bilaga 1 för spelets kod.

5. Genomförande

Vi började vårt projekt med att rita ett kopplingsschema och sätta ihop skärmen och processorn enligt detta. När de var kopplade satte vi in JTAG:en och skrev ett testprogram för att försöka skriva till skärmen. Det tog en hel del tid att förstå hur detta skulle göras och hur man skrev kommandon till processorn så att den läste av på rätt ställe och skrev till rätt ställe på skärmen. Det var då vi lyckats skriva till skärmen som vi insåg att vår ursprungliga idé inte gick att genomföra (se 6.3).

Vi fortsatte istället med Peg Solitaire och kopplade in knappsatsen till processorn och OR-portarna. Vi fick inte knapparna att fungera förrän vi kopplade dem till varsin resistor så att de blev aktiva höga, dvs. skrev en etta endast då en knapp var nedtryckt. I samma veva fick vi även byta ut vår processor eftersom den inte verkade fungera som den skulle.

När väl all hårdvara var igång och fungerade som vi ville gick det snabbt att skriva koden till spelet och börja testa den.

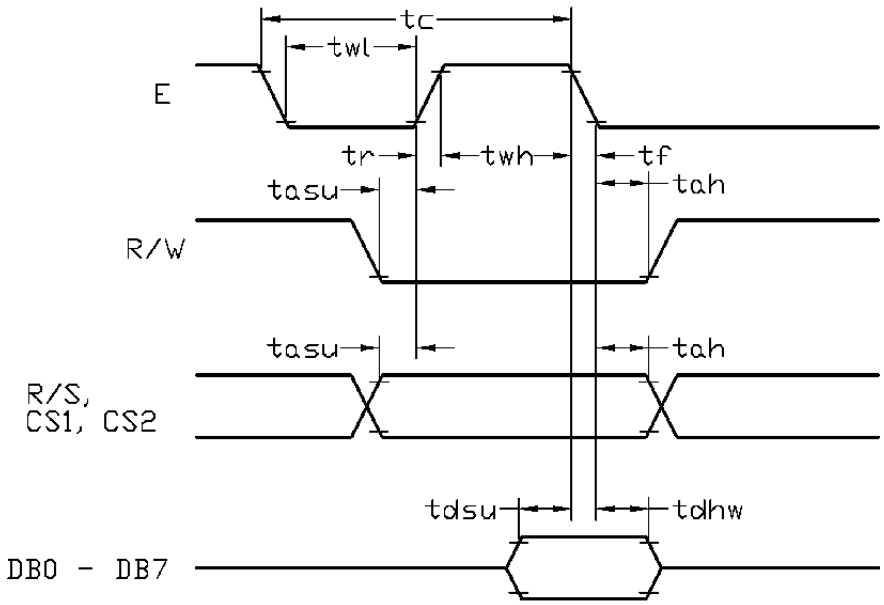
6. Svårigheter

6.1 Programmera i C

Vi hade ingen tidigare erfarenhet av att programmera i C så vi fick läsa oss till hur man skrev metoder och definitioner. Det var också ovant för oss att inte kunna skriva klasser på samma sätt som i Java.

6.2 Timing

När vi började testa skärmen fick vi det inte att fungera tills vi skrev en separat rutin för att aktivera rätt port vid rätt tillfälle när vi ville skriva till skärmen. Rutinen följer schemat nedan.



Host Write Timing Diagram

6.3 Skärmen

Från början var det tänkt att vi skulle göra en Sudoku-lösare. Denna skulle rita upp ett sudoku och användaren skulle kunna fylla i siffror och sedan trycka på en solve-knapp för att lösa sudokut. Idéen var att skriva siffrorna om 6*8 pixlar inklusive rutnät, för att få plats på skärmen. Vi insåg att detta skulle bli väldigt krångligt då vi hade fått skärmen att fungera eftersom vi tänkt skriva 6 pixlar i x-led, men dessa skrevs 8 pixlar på en gång. Vi fick därför byta spel till ett Peg Solitaire, 8*8 pixlar men 7*7 stort, som fick plats på skärmen.

Vi fick även många avbrott från skärmen som dock försvann när vi bytta ut kabeln mellan skärmen och processorn.

6.4 JTAG

Vi hade mycket problem i början med avbrott från JTAG:en och konstruktionen är mycket känslig för om man stöter till den.

6.5 Datablad

Eftersom vi inte tidigare har arbetat med elektroniska komponenter hade vi ingen erfarenhet av att läsa datablad. Detta var ett bekymmer under hela arbetet med projektet och vi fick fråga vår handledare om många detaljer.

7. Resultat

Vår konstruktion fungerar som det är tänkt och det går att spela Peg Solitaire med en markör som kan flyttas över spelplanen och flytta på kulorna. Vi har inte lagt till någon tidtagare eller high-score funktion.

8. Utvärderande diskussion

Vi är mycket nöjda med vår konstruktion. I början hade vi svårt att se hur vi skulle klara av att slutföra projektet då vi tyckte att vi saknade många viktiga grundkunskaper. Vi hade mycket hjälp av kursen Datorteknik så att vi i alla fall hade en förståelse för hur mikroprocessorn fungerade på maskinnivå.

Vi rekommenderar kursen för andra I-are om de har vissa förkunskaper inom Datorteknik. Önskvärt hade varit en djupare genomgång om hur ATMega16 fungerade innan vi startade med projekten.

Slutomdömet är att det har varit roligt och lärorikt. Speciellt att arbeta med hårdvaran var nytt och spännande. Vi fick också oväntade kunskaper i C-programmering som vi kan ha nytta av i framtiden.

9. Referenslista

[1] B.W Kernighan, D.M. Ritchie, The C Programming Language, Second Edition, Prentice-Hall PTR 2007

[2] ATMega16

[3] AVR Beginners, http://www.avrbeginners.net/

10. Appendix

10.1 Bilaga 1

10.1.1 buttons.c

```
#include "buttons.h"
#include "lcdscreen.h"
#include <avr/io.h>
#include "font.h"
#include "digpiSolitaire.h"

/*
* Initialize Buttons
*/
void init_buttons()
{
	BUTTON_CTRL_DDR = 0xF0;
	INTERRUPT_CTRL_DDR = 0x00;
}
/*
* Loop buttons by sending signals to keypad
*/
void loop_buttons()
{
	row = 0;
	while(check_interrupt() == 0)
	{
		if(row == 4)
			row = 0;
		row += 1;
		write_button();
	}
	check_buttons();
}
/*
* Sends signals to the keypad
*/
void write_button()
{
	CLR_BUTTONS;
	switch(row)
	{
	case 1:
	SET_ROW_1;
	break;
	case 2:
	SET_ROW_2;
```

```
		break;
		case 3:
		SET_ROW_3;
		break;
		case 4:
		SET_ROW_4;
		break;
		default:
		break;
		}
}
/*
* Checks which button is pressed calls method accordingly
* J: Jump to
* W: Winner test
* RST: Reset game
* M: Mark peg
* ARROW LEFT: Move left
* ARROW RIGHT: Move right
* ARROW DOWN: Move down
* ARROW UP: Move up
* Buttons 2-9 not in use
*/
void check_buttons()
{
		unsigned char column = (BUTTON_CTRL_PIN & 0x0F);

		switch(row)
		{
		case 1:
		if(column == 0x01)
		{
			reset_game();//RST
			break;
					else if(column== 0x02)
			{
			mark_peg();//M
			break;
		}else if(column == 0x04)
		{
			jump_to(); //J
			break;
		}else if(column == 0x08)
		{                                                                move_left(); //arrow left
			break;
		}
		case 2:
		if(column == 0x08)
```

```
		{
			move_right(); //arrow right;
			break;
		}else
			break;
		case 3:
		if(column == 0x08)
		{
			move_down(); //arrow down
			break;
		}else
			break;
		case 4:
		if(column == 0x08)
		{
			move_up(); //arrow up
			break;
		}else if(column == 0x01)
		{
			winner_test();
			break;
		}
		else
			break;
		default:
			break;
		}
}

/*
* Checks if a buttons is pressed. Returns 1 if pressed, 0 if not
*/
unsigned char check_interrupt()
{
		if(INTERRUPT_CTRL_PIN & 0x01)
		{
			return 1;
		}else
		{
			return 0;
		}
}

/*
*  Prints on the screen as if the game was completed.
*/
void winner_test()
{
```

```
	for(j = 0; j < 49; j++){
		if(board[j] != -1)
		{
			board[j] = 0;
			write_no_peg(j/7, j%7*8+4);
		}
	}
	board[24] = 1;
	x_pos = 24/7;
	y_pos = 24%7;
	write_as_position(x_pos, y_pos*8+4);
}

/*
*  Resets game;
*/
void reset_game()
{
	setup();
}

/*
* Marks position in the matrix and shows peg as marked on the screen
*/
void mark_peg()
{
	if(marked != -1)
		{
			if(board[marked] == 1)
				write_full_peg(marked/7, (marked%7)*8+4);
			else if(board[marked] == 0)
				write_no_peg(marked/7, (marked%7)*8+4);
			else
				write_empty(marked/7, (marked%7)*8+4);
		}
		marked = x_pos*7+y_pos;
		write_as_marked(x_pos, y_pos*8+4);
}

/*
* Jumps to chosen position and prints the result of the move on the screen
*/
void jump_to()
{
		remove_peg();
		write_as_position(x_pos, y_pos*8+4);
}
```

```
/*
* Moves postition one step to the left
* and prints the result of the move on the screen
*/
void move_left()
{
		p = x_pos*7+y_pos;
		if(board[p] == 1 && marked != p)
			write_full_peg(x_pos, y_pos*8+4);
		else if(board[p] == 0 && marked != p)
			write_no_peg(x_pos, y_pos*8+4);
		else if(board[p] == -1)
			write_empty(x_pos, y_pos*8+4);
		if(y_pos == 0)
			y_pos = 6;
		else
			y_pos--;
		if(marked != x_pos*7+y_pos)
			write_as_position(x_pos, y_pos*8+4);
}

/*
* Moves postition one step to the right
* and prints the result of the move on the screen
*/
void move_right()
{
		p = x_pos*7+y_pos;
		if(board[p] == 1 && marked != p)
			write_full_peg(x_pos, y_pos*8+4);
		else if(board[p] == 0 && marked != p)
			write_no_peg(x_pos, y_pos*8+4);
		else if(board[p] == -1)
			write_empty(x_pos, y_pos*8+4);
		if(y_pos == 6)
			y_pos = 0;
		else
			y_pos++;
		if(marked != x_pos*7+y_pos)
			write_as_position(x_pos, y_pos*8+4);
}

/*
* Moves postition one step up
* and prints the result of the move on the screen
*/
void move_up()
{
```

```
		p = x_pos*7+y_pos;
		if(board[p] == 1 && marked != p)
			write_full_peg(x_pos, y_pos*8+4);
		else if(board[p] == 0 && marked != p)
			write_no_peg(x_pos, y_pos*8+4);
		else if(board[p] == -1)
			write_empty(x_pos, y_pos*8+4);
		if(x_pos == 0)
			x_pos = 6;
		else
			x_pos--;
		if(marked != x_pos*7+y_pos)
			write_as_position(x_pos, y_pos*8+4);
}

/*
* Moves postition one step down
* and prints the result of the move on the screen
*/
void move_down()
{
		p = x_pos*7+y_pos;
		if(board[p] == 1 && marked != p)
			write_full_peg(x_pos, y_pos*8+4);
		else if(board[p] == 0 && marked != p)
			write_no_peg(x_pos, y_pos*8+4);
		else if(board[p] == -1)
			write_empty(x_pos, y_pos*8+4);
		if(x_pos == 6)
			x_pos = 0;
		else
			x_pos++;
		if(marked != x_pos*7+y_pos)
			write_as_position(x_pos, y_pos*8+4);
}
```

10.1.2 lcdscreen.c

```
#include <avr/io.h>
#include "lcdscreen.h"
#include "font.h"
#include "digpiSolitaire.h"
/*
* Enable E signal to display for timing.
*/
void e_strobe(void)
{
```

```
	SET_EN;
	CLR_EN;
}
/*
* Starts up lcd-screen.
*/
void lcdInit(void)
{
	LCD_DATA_DDR = 0xFF;
	LCD_CTRL_DDR = 0xFF;
	CLR_RST;
	SET_RST;
}
/*
* Turns lcd-screen on.
*/
void lcdOn(void)
{
	LCD_CS0;
	lcdWriteCmd(DISPLAY_ON_CMD | ON);
	LCD_CS1;
	lcdWriteCmd(DISPLAY_ON_CMD | ON);
	LCD_NOCS;
}
/*
* Routine for writing comand to the screen.
*/
void lcdWriteCmd(unsigned char cmd)
{
	CLR_RS;
	CLR_RW;
	LCD_DATA_DDR = 0xFF;
	LCD_DATA_PORT = cmd;
	e_strobe();
	SET_RW;
}
/*
* Routine for writing data to the screen.
*/
void lcdWriteData(unsigned char data)
{
	SET_RS;
	CLR_RW;
	LCD_DATA_DDR = 0xFF;
	LCD_DATA_PORT = data;
	e_strobe();
	SET_RW;
}
```

```
/*
* Prints a hole without peg to the screen to page x, pixel-column y.
*/
void write_no_peg(unsigned char x, unsigned char y)
{
	lcdGoTo(x,y);
	lcdWriteData(no_peg_a);
	lcdWriteData(no_peg_b);
	lcdWriteData(no_peg_c);
	lcdWriteData(no_peg_d);
	lcdWriteData(no_peg_e);
	lcdWriteData(no_peg_f);
	lcdWriteData(no_peg_g);
	lcdWriteData(no_peg_h);

}
/*
*Prints peg to the screen to page x, pixel-column y.
*/
void write_full_peg(unsigned char x, unsigned char y)
{
	lcdGoTo(x,y);
	lcdWriteData(full_peg_a);
	lcdWriteData(full_peg_b);
	lcdWriteData(full_peg_c);
	lcdWriteData(full_peg_d);
	lcdWriteData(full_peg_e);
	lcdWriteData(full_peg_f);
	lcdWriteData(full_peg_g);
	lcdWriteData(full_peg_h);

}
/*
*Checks if there is a peg or not at the position an calls according methods
* for printing position square.
*/
void write_as_position(unsigned char x, unsigned char y)
{
	int pos = x_pos*7+y_pos;
	lcdGoTo(x,y);
	if(board[pos] == 1)
		write_as_pos_full();
	else if(board[pos] == 0)
		write_as_pos_no();
	else
		write_as_pos();
}
/*
```

```
*Prints position square for peg to the screen at page x, pixel-column y.
*/
void write_as_pos_full()
{
	lcdWriteData(full_pos_peg_a);
	lcdWriteData(full_pos_peg_b);
	lcdWriteData(full_pos_peg_c);
	lcdWriteData(full_pos_peg_d);
	lcdWriteData(full_pos_peg_e);
	lcdWriteData(full_pos_peg_f);
	lcdWriteData(full_pos_peg_g);
	lcdWriteData(full_pos_peg_h);
}
/*
*Prints position square for hole without peg to the screen at page x, pixel-column y.
*/
void write_as_pos_no()
{
	lcdWriteData(no_pos_peg_a);
	lcdWriteData(no_pos_peg_b);
	lcdWriteData(no_pos_peg_c);
	lcdWriteData(no_pos_peg_d);
	lcdWriteData(no_pos_peg_e);
	lcdWriteData(no_pos_peg_f);
	lcdWriteData(no_pos_peg_g);
	lcdWriteData(no_pos_peg_h);
}
/*
*Prints position square for empty space to the screen at page x, pixel-column y.
*/
void write_as_pos()
{
	lcdWriteData(pos_peg_a);
	lcdWriteData(pos_peg_b);
	lcdWriteData(pos_peg_c);
	lcdWriteData(pos_peg_d);
	lcdWriteData(pos_peg_e);
	lcdWriteData(pos_peg_f);
	lcdWriteData(pos_peg_g);
	lcdWriteData(pos_peg_h);
}
/*
*Checks if there is a peg or not at the position an calls according methods
* for printing marked square. Goes to page x, pixel column y.
*/
void write_as_marked(unsigned char x, unsigned char y)
{
	int pos = x_pos*7+y_pos;
```

```
	lcdGoTo(x,y);
	if(board[pos] == 1)
		write_as_m_full();
	else if(board[pos] == 0)
		write_as_m_no();
	else
		write_as_m();
}
/*
*Prints marked square for peg to the screen.
*/
void write_as_m_full()
{
	lcdWriteData(full_m_peg_a);
	lcdWriteData(full_m_peg_b);
	lcdWriteData(full_m_peg_c);
	lcdWriteData(full_m_peg_d);
	lcdWriteData(full_m_peg_e);
	lcdWriteData(full_m_peg_f);
	lcdWriteData(full_m_peg_g);
	lcdWriteData(full_m_peg_h);
}
/*
*Prints marked square for hole without peg to the screen.
*/
void write_as_m_no()
{
	lcdWriteData(no_m_peg_a);
	lcdWriteData(no_m_peg_b);
	lcdWriteData(no_m_peg_c);
	lcdWriteData(no_m_peg_d);
	lcdWriteData(no_m_peg_e);
	lcdWriteData(no_m_peg_f);
	lcdWriteData(no_m_peg_g);
	lcdWriteData(no_m_peg_h);
}
/*
*Prints marked square for empty space without peg to the screen.
*/
void write_as_m()
{
	lcdWriteData(m_peg_a);
	lcdWriteData(m_peg_b);
	lcdWriteData(m_peg_c);
	lcdWriteData(m_peg_d);
	lcdWriteData(m_peg_e);
	lcdWriteData(m_peg_f);
	lcdWriteData(m_peg_g);
```

```
	lcdWriteData(m_peg_h);
}
/*
*Prints empty space to the screen at page x, pixel-column y.
*/
void write_empty(unsigned char x, unsigned char y)
{
	lcdGoTo(x,y);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);
	lcdWriteData(empty);

}
/*
*Prints winning text to the screen at page x, pixel-column y.
*/
void write_letter(unsigned char x, unsigned char y)
{
	lcdGoTo(x,y);

		lcdWriteData(l_y_a);
		lcdWriteData(l_y_b);
		lcdWriteData(l_y_c);
		lcdWriteData(l_y_d);
		lcdWriteData(l_y_e);
		lcdWriteData(l_y_f);
		lcdWriteData(l_y_g);
		lcdWriteData(l_y_h);

		lcdWriteData(l_o_a);
		lcdWriteData(l_o_b);
		lcdWriteData(l_o_c);
		lcdWriteData(l_o_d);
		lcdWriteData(l_o_e);
		lcdWriteData(l_o_f);
		lcdWriteData(l_o_g);
		lcdWriteData(l_o_h);

		lcdWriteData(l_u_a);
		lcdWriteData(l_u_b);
		lcdWriteData(l_u_c);
		lcdWriteData(l_u_d);
		lcdWriteData(l_u_e);
```

```
                lcdWriteData(l_u_f);
                lcdWriteData(l_u_g);
                lcdWriteData(l_u_h);

                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);
                lcdWriteData(empty);

                lcdWriteData(l_w_a);
                lcdWriteData(l_w_b);
                lcdWriteData(l_w_c);
                lcdWriteData(l_w_d);
                lcdWriteData(l_w_e);
                lcdWriteData(l_w_f);
                lcdWriteData(l_w_g);
                lcdWriteData(l_w_h);

                lcdWriteData(l_i_a);
                lcdWriteData(l_i_b);
                lcdWriteData(l_i_c);
                lcdWriteData(l_i_d);
                lcdWriteData(l_i_e);
                lcdWriteData(l_i_f);
                lcdWriteData(l_i_g);
                lcdWriteData(l_i_h);

                lcdWriteData(l_n_a);
                lcdWriteData(l_n_b);
                lcdWriteData(l_n_c);
                lcdWriteData(l_n_d);
                lcdWriteData(l_n_e);
                lcdWriteData(l_n_f);
                lcdWriteData(l_n_g);
                lcdWriteData(l_n_h);

                LCD_NOCS;
}
/*
*Sets the screen position to page x, pixel-column y.
*/
void lcdGoTo(unsigned char x, unsigned char y)
{
        if(y > 63)
```

```
	{
	LCD_CS1;
	y -= 64;
	}else
	{
	LCD_CS0;
	}
	lcdWriteCmd(DISPLAY_SET_X | x);
	lcdWriteCmd(DISPLAY_SET_Y | y);
}
/*
*Clears lcd screen.
*/
void lcdCls(void)
{
	unsigned char x, y;
	for (x = 0; x < 8; x++)
	{
		for (y = 0; y < 128; y++)
		{
			lcdGoTo(x,y);
			lcdWriteData(0x00);
		}
		lcdGoTo(0,0);
	}
}
```

10.1.3 digpiSolitaire.c

```
#include "digpiSolitaire.h"
#include "lcdscreen.h"
#include <avr/io.h>
#include "buttons.h"

int main(void)
{
	setup();
	while(1)
	{
		start_game();
	}
}
/*
* Sets up lcd screen, buttons and matrix
*/
void setup()
```

```
{
	lcdInit();
	lcdOn();
	init_buttons();
	lcdCls();
	setup_board();
}
/*
* Sets up matrix and prints start-up board on the screen
*/
void setup_board()
{
	k = 1;
	board[0] = -1;
	while(k < 49)
	{
		x = k / 7;
		y = k % 7;
		if(x < 2 || x > 4)
		{
			if(y < 2 || y > 4)
				board[k] = -1;
			else
			{
				board[k] = 1;
				write_full_peg(x, y*8+4);
			}
		}
		else
		{
			board[k] = 1;
			write_full_peg(x, y*8+4);
		}
		k++;
	}

	board[24] = 0;
	x_pos = 24/7;
	y_pos = 24%7;
	write_as_position(x_pos, y_pos*8+4);
	marked = -1;
}
/*
* Runs game
*/
void start_game()
{
	while(check_board())
```

```
	{
	loop_buttons();
	}
	print_win();
	board[24] = 0;
}
/*
* Performes the move if possible according to Solitaire rules.
*/
void remove_peg(){

	k = x_pos*7+y_pos;
	if(k == (marked+14) && board[marked + 14] == 0) //flyttat ner
	{
		if(board[k-7] == 1)
		{
			board[k-7] = 0;
			board[k] = 1;
			board[marked] = 0;
			write_no_peg(x_pos-1, (y_pos*8+4));
			write_full_peg(x_pos, (y_pos*8+4));
			write_no_peg(x_pos-2, (y_pos*8+4));
			marked = -1;
		}
	}else if(k == (marked + 2) && board[marked +2] == 0) //flyttat till höger
	{
		if(board[k-1] == 1)
		{
			board[k-1] = 0;
			board[k] = 1;
			board[marked] = 0;
			write_no_peg(x_pos, (y_pos*8-4));
			write_full_peg(x_pos, (y_pos*8+4));
			write_no_peg(x_pos, (y_pos*8-12));
			marked = -1;
		}
	}else if(k == (marked - 2) && board[marked -2] == 0) //flyttat till vänster
	{
		if(board[k+1] == 1)
		{
			board[k+1] = 0;
			board[k] = 1;
			board[marked] = 0;
			write_no_peg(x_pos, (y_pos*8+12));
			write_full_peg(x_pos, (y_pos*8+4));
			write_no_peg(x_pos, (y_pos*8+20));
			marked = -1;
		}
```

```
	}else if(k ==(marked - 14) && board[marked -14] == 0) //flyttat upp
	{
		if(board[k+7] == 1)
		{
			board[k+7] = 0;
			board[k] = 1;
			board[marked] = 0;
			write_no_peg(x_pos+1, (y_pos*8+4));
			write_full_peg(x_pos, (y_pos*8+4));
			write_no_peg(x_pos+2, (y_pos*8+4));
			marked = -1;
		}
	}
}
/*
* Checks if the game is completed.
*/
int check_board()
{
	int i;
	int count = 0;

	for(i = 0; i < 49; i++)
	{
		if(board[i] == 1)
			count++;
	}
	if(count == 1)
		return 0;
	else
		return 1;
}
/*
* Prints winning text.
*/
void print_win(){

	write_letter(3, 65);

}
```

10.2 Bilaga 2: h-filer

10.2.1 buttons.h

```
/*
* Define Port D
*/
#define BUTTON_CTRL_PORT PORTD
#define BUTTON_CTRL_PIN PIND
#define BUTTON_CTRL_DDR DDRD
/*
*Define Port C
*/
#define INTERRUPT_CTRL_PORT PORTC
#define INTERRUPT_CTRL_PIN PINC
#define INTERRUPT_CTRL_DDR DDRC

/*
*Define  Columns
*/
#define COL_1 (PIND & 0x01)
#define COL_2 (PIND & 0x02)
#define COL_3 (PIND & 0x04)
#define COL_4 (PIND & 0x08)
/*
*Define  Rows, set and clear
*/
#define SET_ROW_1 (BUTTON_CTRL_PORT |= (1 << PD7))
#define CLR_ROW_1 (BUTTON_CTRL_PORT &= ~(1 << PD7))
#define SET_ROW_2 (BUTTON_CTRL_PORT |= (1 << PD6))
#define CLR_ROW_2 (BUTTON_CTRL_PORT &= ~(1 << PD6))
#define SET_ROW_3 (BUTTON_CTRL_PORT |= (1 << PD5))
#define CLR_ROW_3 (BUTTON_CTRL_PORT &= ~(1 << PD5))
#define SET_ROW_4 (BUTTON_CTRL_PORT |= (1 << PD4))
#define CLR_ROW_4 (BUTTON_CTRL_PORT &= ~(1 << PD4))
#define CLR_BUTTONS CLR_ROW_1; CLR_ROW_2; CLR_ROW_3; CLR_ROW_4;
#define BUTTON_INTERRUPT (INTERRUPT_CTRL_PIN & 0x01)

/*
*Methods
*/
void init_buttons();
void check_buttons();
void write_button();
unsigned char check_interrupt();
void loop_buttons();
void jump_to();
void mark_peg();
void reset_game();
```

```
void move_up();
void move_down();
void move_left();
void move_right();
void winner_test();

/*
*Global variables
*/
unsigned char row;
int p,j;
```

10.2.2 lcdscreen.h

```
/*
* Define Port A for Databus
*/
#define LCD_DATA_PORT PORTA //all a-ports
#define LCD_DATA_PIN PINA
#define LCD_DATA_DDR DDRA //busy-flag
/*
* Define Port B for control signals
*/
#define LCD_CTRL_PORT PORTB //all b-ports
#define LCD_CTRL_PIN PINB
#define LCD_CTRL_DDR DDRB //busy-flag
/*
* Define control signals to LCD-Display
*/
#define LCD_CS1P PB0
#define LCD_CS2P PB1
#define LCD_RST PB2
#define LCD_RW PB3
#define LCD_RS PB4
#define LCD_E PB5
/*
* Define
*/
#define SET_CS1 (LCD_CTRL_PORT |= (1 << LCD_CS1P))
#define CLR_CS1 (LCD_CTRL_PORT &= ~(1 << LCD_CS1P))
#define SET_CS2 (LCD_CTRL_PORT |= (1 << LCD_CS2P))
#define CLR_CS2 (LCD_CTRL_PORT &= ~(1 << LCD_CS2P))
#define SET_RST (LCD_CTRL_PORT |= (1 << LCD_RST))
#define CLR_RST (LCD_CTRL_PORT &= ~(1 << LCD_RST))
#define SET_RW (LCD_CTRL_PORT |= (1 << LCD_RW))
#define CLR_RW (LCD_CTRL_PORT &= ~(1 << LCD_RW))
#define SET_RS (LCD_CTRL_PORT |= (1 << LCD_RS))
#define CLR_RS (LCD_CTRL_PORT &= ~(1 << LCD_RS))
```

```
#define SET_EN (LCD_CTRL_PORT |= (1 << LCD_E))
#define CLR_EN (LCD_CTRL_PORT &= ~(1 << LCD_E))
#define LCD_CS0 SET_CS1;CLR_CS2;
#define LCD_CS1 CLR_CS1;SET_CS2;
#define LCD_NOCS CLR_CS1;CLR_CS2;
#define DISPLAY_STATUS_BUSY 0x80
#define DISPLAY_SET_X 0xB8
#define DISPLAY_SET_Y 0x40
#define DISPLAY_START_LINE 0xC0
#define DISPLAY_ON_CMD 0x3E
#define ON 0x01
/*
* Methods
*/
void lcdWriteData(unsigned char);
void lcdWriteCmd(unsigned char);
void lcdInit();
void lcdOn();
void lcdCls();
void write_no_peg(unsigned char, unsigned char);
void write_full_peg(unsigned char, unsigned char);
void write_as_position(unsigned char, unsigned char);
void write_as_pos_no();
void write_as_pos_full();
void write_as_pos();
void write_as_marked(unsigned char, unsigned char);
void write_as_m_no();
void write_as_m_full();
void write_as_m();
void write_empty(unsigned char, unsigned char);
void write_letter(unsigned char, unsigned char);
void lcdGoTo(unsigned char, unsigned char);
```

10.2.3 digpiSolitaire.h

```
/*
*Global variables
*/
int board[49];
int marked;
unsigned char x_pos;
unsigned char y_pos;
int check_board();
int k,x,y;

/*
*Methods
*/
```

```
void setup();
void setup_board();
void start_game();
void remove_peg();
void print_win();
```

10.2.4 font.h

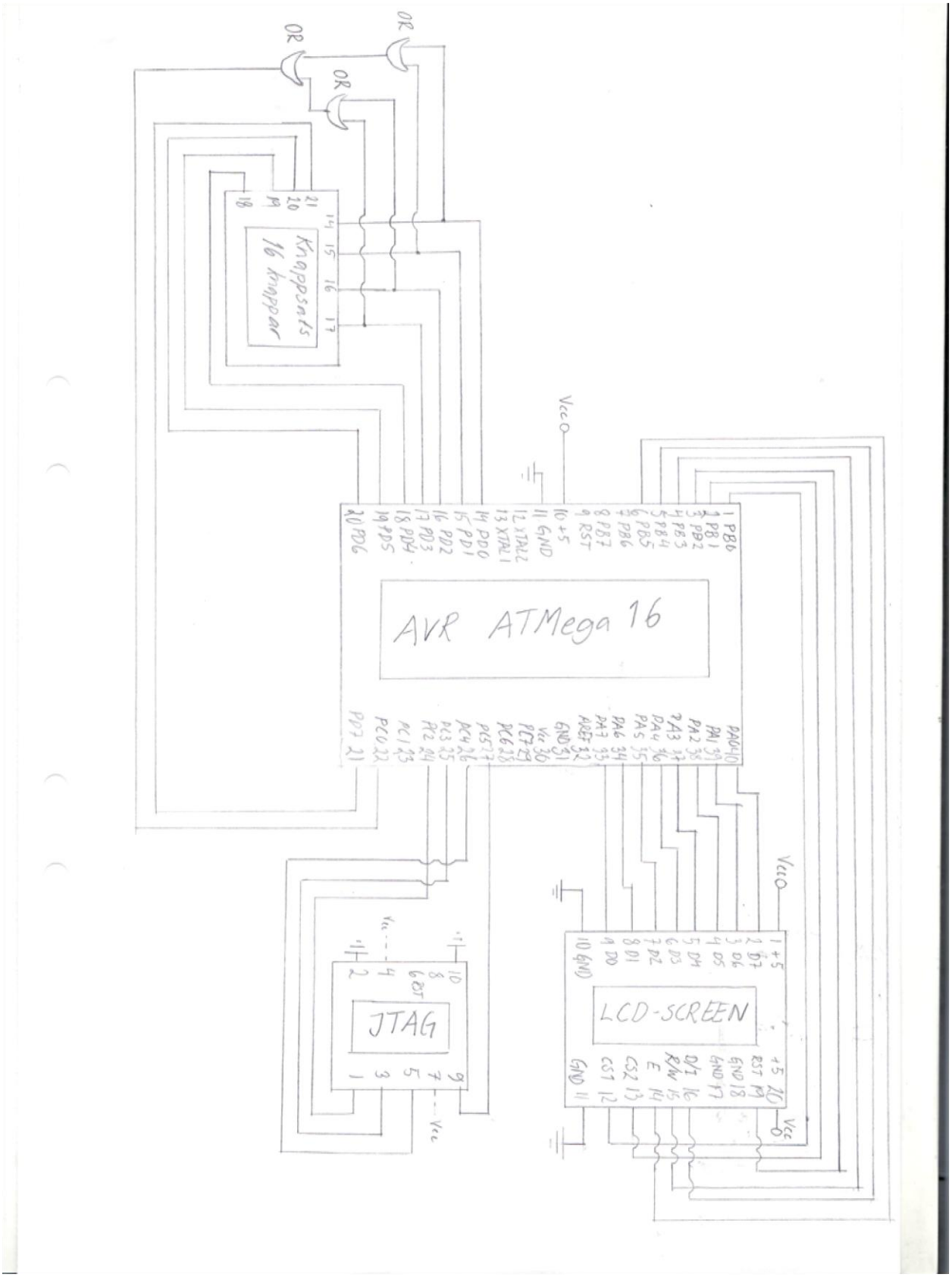
```
//empty hole
#define no_peg_a 0x00
#define no_peg_b 0x00
#define no_peg_c 0x18
#define no_peg_d 0x24
#define no_peg_e 0x24
#define no_peg_f 0x18
#define no_peg_g 0x00
#define no_peg_h 0x00

//peg
#define full_peg_a 0x00
#define full_peg_b 0x00
#define full_peg_c 0x18
#define full_peg_d 0x3C
#define full_peg_e 0x3C
#define full_peg_f 0x18
#define full_peg_g 0x00
#define full_peg_h 0x00

//position at hole
#define no_pos_peg_a 0xAA
#define no_pos_peg_b 0x01
#define no_pos_peg_c 0x98
#define no_pos_peg_d 0x25
#define no_pos_peg_e 0xA4
#define no_pos_peg_f 0x19
#define no_pos_peg_g 0x80
#define no_pos_peg_h 0x55

//position at peg
#define full_pos_peg_a 0xAA
#define full_pos_peg_b 0x01
#define full_pos_peg_c 0x98
#define full_pos_peg_d 0x3D
#define full_pos_peg_e 0xBC
#define full_pos_peg_f 0x19
#define full_pos_peg_g 0x80
#define full_pos_peg_h 0x55
```

```
//Position at empty space
#define pos_peg_a 0xAA
#define pos_peg_b 0x01
#define pos_peg_c 0x80
#define pos_peg_d 0x01
#define pos_peg_e 0x80
#define pos_peg_f 0x01
#define pos_peg_g 0x80
#define pos_peg_h 0x55

//Marked at hole
#define no_m_peg_a 0xFF
#define no_m_peg_b 0x81
#define no_m_peg_c 0x99
#define no_m_peg_d 0xA5
#define no_m_peg_e 0xA5
#define no_m_peg_f 0x99
#define no_m_peg_g 0x81
#define no_m_peg_h 0xFF

//Marked at peg
#define full_m_peg_a 0xFF
#define full_m_peg_b 0x81
#define full_m_peg_c 0x99
#define full_m_peg_d 0xBD
#define full_m_peg_e 0xBD
#define full_m_peg_f 0x99
#define full_m_peg_g 0x81
#define full_m_peg_h 0xFF

Marked at empty space
#define m_peg_a 0xFF
#define m_peg_b 0x81
#define m_peg_c 0x81
#define m_peg_d 0x81
#define m_peg_e 0x81
#define m_peg_f 0x81
#define m_peg_g 0x81
#define m_peg_h 0xFF

//Empty space
#define empty 0x00

//Letter Y
#define l_y_a 0x00
#define l_y_b 0x00
#define l_y_c 0x06
#define l_y_d 0x08
```

```
#define l_y_e 0xF0
#define l_y_f 0x08
#define l_y_g 0x06
#define l_y_h 0x00
//Letter O
#define l_o_a 0x00
#define l_o_b 0x38
#define l_o_c 0x44
#define l_o_d 0x82
#define l_o_e 0x82
#define l_o_f 0x44
#define l_o_g 0x38
#define l_o_h 0x00

//Letter U
#define l_u_a 0x00
#define l_u_b 0x7E
#define l_u_c 0x80
#define l_u_d 0x80
#define l_u_e 0x80
#define l_u_f 0x80
#define l_u_g 0x7E
#define l_u_h 0x00

//Letter W
#define l_w_a 0x00
#define l_w_b 0x00
#define l_w_c 0x7E
#define l_w_d 0x80
#define l_w_e 0x70
#define l_w_f 0x80
#define l_w_g 0x7E
#define l_w_h 0x00

//Letter I
#define l_i_a 0x00
#define l_i_b 0x00
#define l_i_c 0x00
#define l_i_d 0x82
#define l_i_e 0xFE
#define l_i_f 0x82
#define l_i_g 0x00
#define l_i_h 0x00

//Letter N
#define l_n_a 0x00
#define l_n_b 0xFE
#define l_n_c 0x04
```

```
#define l_n_d 0x08
#define l_n_e 0x10
#define l_n_f 0x20
#define l_n_g 0xFE
#define l_n_h 0x00
```

10.3 Bilaga 3: Kopplingsschema



10.4 Bilaga 4: Knappsats